# [What's New in C# 6](https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-6)

<https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-6>

* Read-only Auto-properties:

You can create read-only auto-properties that can be set only in constructors. Public string FirstName { get ; }

* Auto-Property Initializers:

You can write initialization expressions to set the initial value of an auto-property. public ICollection<double> Grades { get ; } = new List<double>();

* Expression-bodied function members:

You can author one-line methods using lambda expressions.

* using static: using static System.Console;

You can import all the methods of a single class into the current namespace. Using static System.Math;

* Null - conditional operators: person?.FirstName

You can concisely and safely access members of an object while still checking for null with the null conditional operator.

* String Interpolation: $”{FirstName} {LastName}”

You can write string formatting expressions using inline expressions instead of positional arguments.

* Exception filters:   
  catch (HttpRequestException e) when ( e.Message.Contains(“301”))

You can catch expressions based on properties of the exception or other program state.

* nameof Expressions: nameof(lastName)

You can let the compiler generate string representations of symbols.

* await in catch and finally blocks:

You can use await expressions in locations that previously disallowed them.

* index initializers:

private Dictionary<int, string> webErrors = new Dictionary<int, string>

{

[404] = "Page not Found"

};

* Extension methods for collection initializers:

Collection initializers can rely on accessible extension methods, in addition to member methods.

* Improved overload resolution:

Some constructs that previously generated ambiguous method calls now resolve correctly.

# [What's new in C# 7](https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-7)

<https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-7>

C# 7 adds a number of new features to the C# language:

* out variables: int.TryParse(input, out int result)

You can declare out values inline as arguments to the method where they are used.

* Tuples

You can create lightweight, unnamed types that contain multiple public fields. Compilers and IDE tools understand the semantics of these types.

* Discards

Discards are temporary, write-only variables used in assignments when you don't care about the value assigned. They are particularly useful when deconstructing tuples and user-defined types, as well as when calling methods with out parameters.

* Pattern Matching

You can create branching logic based on arbitrary types and values of the members of those types.

* ref locals and returns

Method arguments and local variables can be references to other storage.

* Local Functions

You can nest functions inside other functions to limit their scope and visibility.

* More expression-bodied members

The list of members that can be authored using expressions has grown. constructors, finalizers, and get and set accessors on properties and indexers.

* throw Expressions

This features enables using throw expressions in initialization expressions

* Generalized async return types

Methods declared with the async modifier can return other types in addition to Task and Task<T>.

* Numeric literal syntax improvements

New tokens improve readability for numeric constants.

public const int Sixteen = 0b0001\_0000;

# [What's new in C# 7.1](https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-7-1)

<https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-7-1>

C# 7.1 is the first point release to the C# language. It marks an increased release cadence for the language. You can use the new features sooner, ideally when each new feature is ready. C# 7.1 adds the ability to configure the compiler to match a specified version of the language. That enables you to separate the decision to upgrade tools from the decision to upgrade language versions.

C# 7.1 adds the language version selection configuration element, three new language features and new compiler behavior.

The new language features in this release are:

* async Main method

static async Task Main()

{

await SomeAsyncMethod();

}

The entry point for an application can have the async modifier.

* default literal expressions

You can use default literal expressions in default value expressions when the target type can be inferred.

old: Func<string, bool> whereClause = default(Func<string, bool>);

new: Func<string, bool> whereClause = default;

* Inferred tuple element names

The names of tuple elements can be inferred from tuple initialization in many cases.

int count = 5;

string label = "Colors used in the map";

var pair = (count, label); // element names are "count" and "label"

* Reference assembly generation (meta-data only assemblies)

Finally, the compiler has two options /refout and /refonly that control reference assembly generation.

Metadata-only assemblies have their method bodies replaced with a single throw null body, but include all members except anonymous types.

# [What's new in C# 7.2](https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-7-2)

<https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-7-2>

C# 7.2 is another point release that adds a number of useful features. One theme for this release is working more efficiently with value types by avoiding unnecessary copies or allocations.

The remaining features are small, nice-to-have features.

C# 7.2 uses the language version selection configuration element to select the compiler language version.

The new language features in this release are:

* Reference semantics with value types
  + In parameter designation
  + ref readonly returns
  + readonly struct type
  + ref struct type

A combination of syntax improvements that enable working with value types using reference semantics. Designed for performance critical algorithms where memory allocations can be critical to achieving the necessary performance.

* Non-trailing named arguments

Named arguments can be followed by positional arguments, as long as named arguments are used in the correct position (C# 7.2)

* Leading underscores in numeric literals

int binaryValue = 0b\_0101\_0101;

Numeric literals can now have leading underscores before any printed digits.

* private protected access modifier

The private protected access modifier enables access for derived classes in the same assembly.

Finally, a new compound access modifier: private protected indicates that a member may be accessed by containing class or derived classes that are declared in the same assembly.

private protected limits access to derived types declared in the same assembly.